CSCI 3753: Operating Systems

Week 10
October 31st, 2025

University of Colorado Boulder

Happy Halloween!

Announcements

- PS3 due Sunday at midnight!
- Quiz 10 due at midnight tonight!
- PA7 is due NEXT Sunday!

- As always recitation materials are stored here

Memory Management

- Operating systems must manage memory
  - memory pages or segments
- Not all memory is utilized at once
  - OS will operate on a subset of memory
- Optimized memory usage
  - Reduced memory access time -> better performance

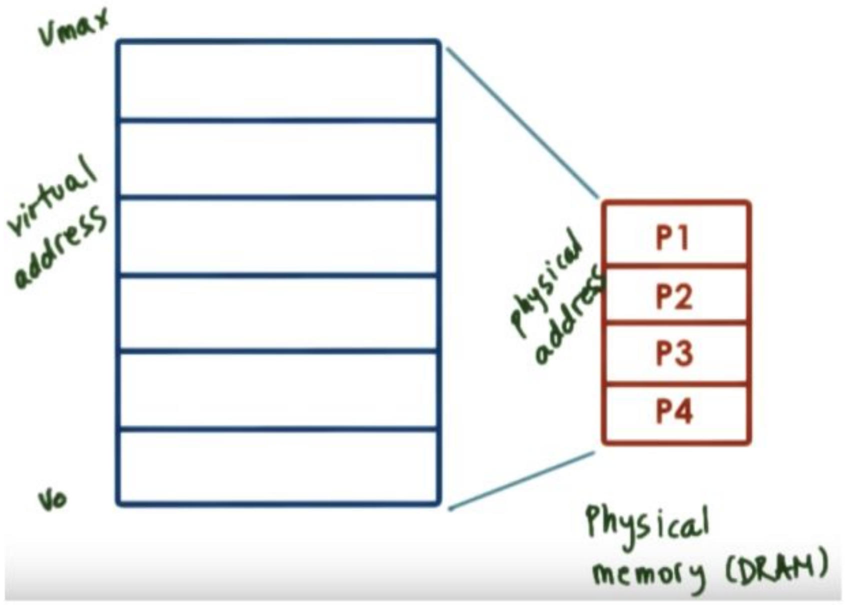
Page Based Memory Management

- Virtual vs Physical Memory

- Keep few pages in memory, rest on disk

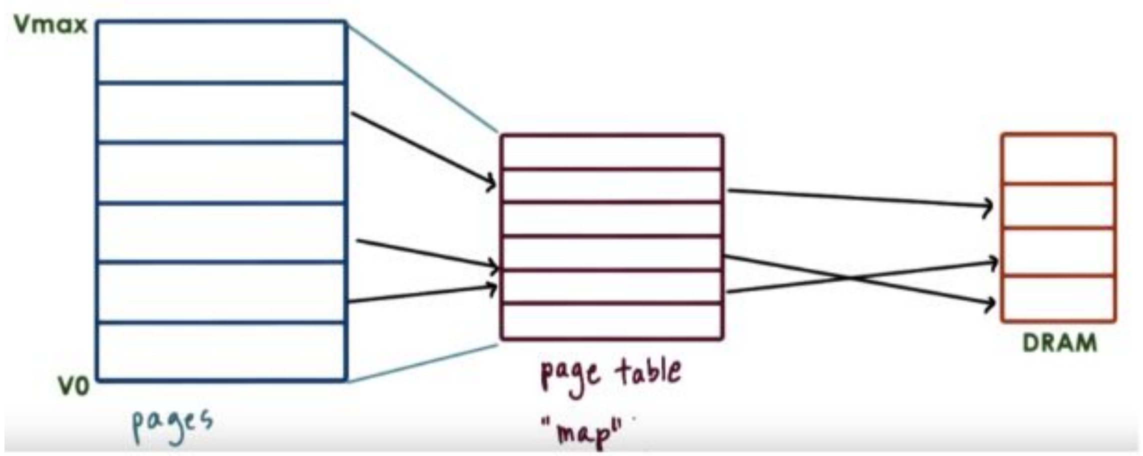
- Allocate
  - allocation, replacement
  - pages -> page frames

- Arbitration
  - Address translation
  - Address validation
  - Page tables

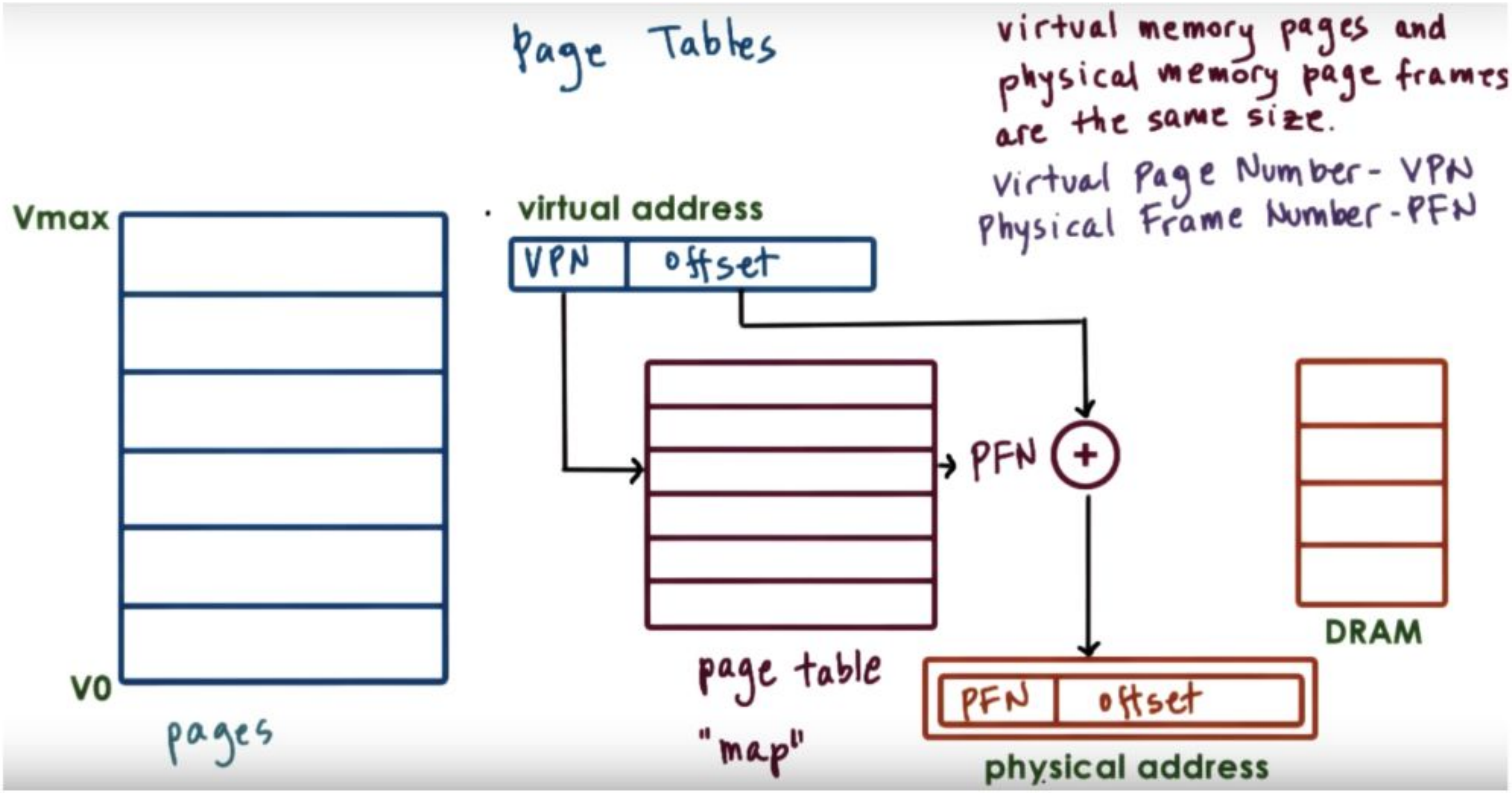


Page Tables

- Virtual memory pages and physical memory page frames are of the same size
- Page table is like a map that tells the OS where to find the virtual memory reference
- Virtual address has an offset



Page Tables


Page Tables
virtual memory pages and physical memory page frames are the same size.
Virtual Page Number - VPN
Physical Frame Number - PFN
Vmax
V0
pages
virtual address
VPN
offset
page table
"map"
PFN
+
DRAM
PFN
offset
physical address

Page Fault

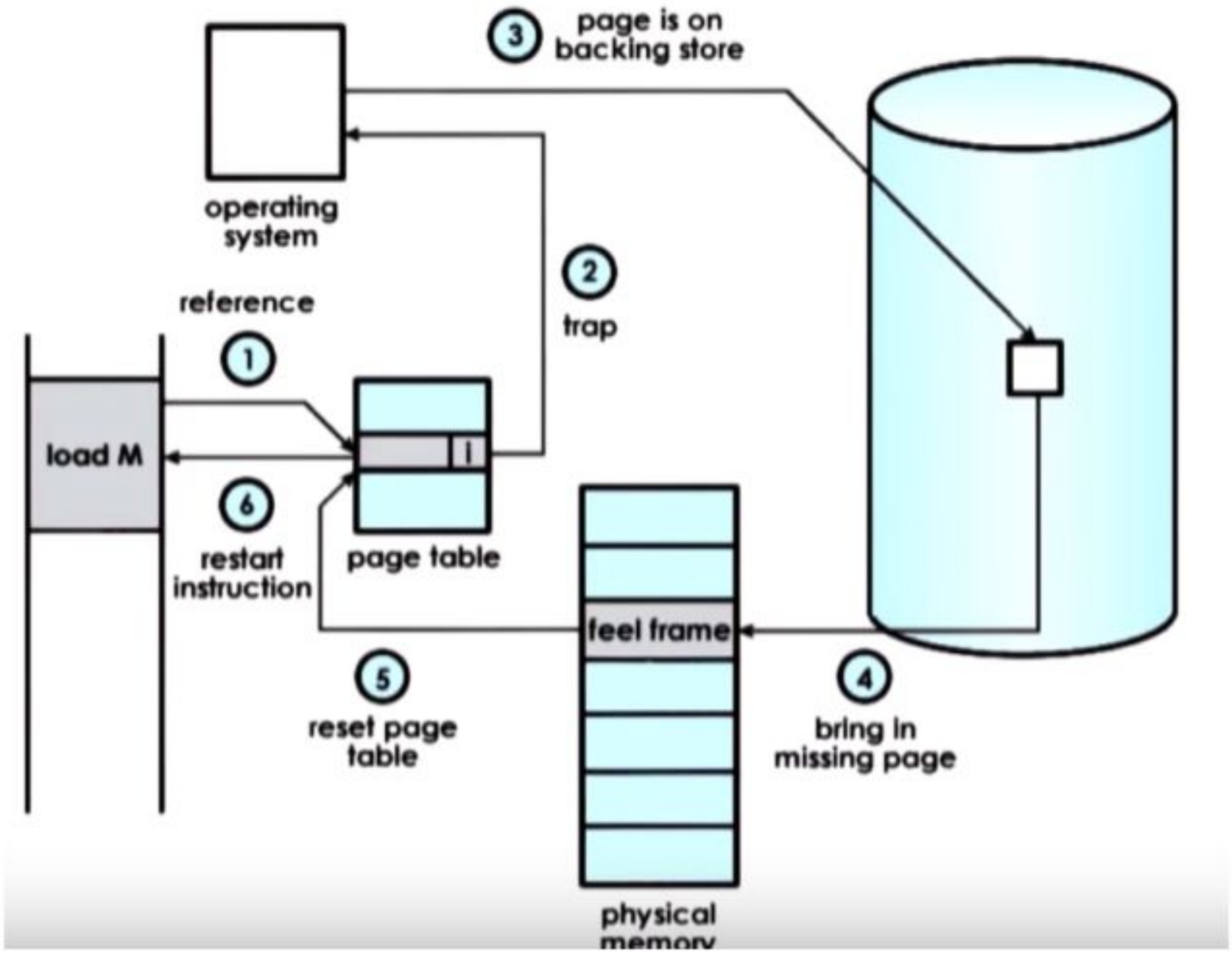
- A referenced page is NOT loaded in memory
- OS blocks the process and retrieves the referenced page
- Significant performance overhead
  - Need to keep page fault frequency low
    - (e.g. less than 1 in 107 for overhead less than 10%)

Demand Paging

- Virtual Memory >> Physical Memory
  - Virtual memory page not always in physical memory
  - Physical page frame can be saved and restored to/from secondary storage

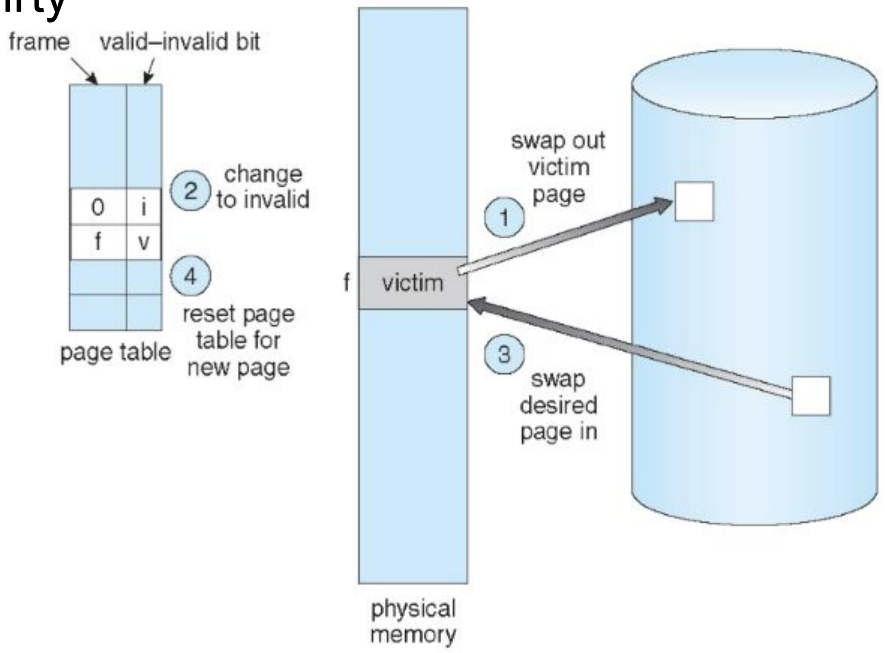
- Demand Paging
  - Pages swapped in and out of memory and swap partition (disk)

Demand Paging



Basic Page Replacement Process

1. Locate the desired page on disk
2. Find a free frame
    a. If there is a free frame, use it
    b. If not, use a page replacement policy to
        i. Select a victim frame
        ii. Write victim frame to disk if dirty



3. Brind the desired page into the newly free frame and update the page and frame tables

4. Continue the process by restarting the instruction that caused the trap

Page Replacement

- Virtual Memory address space >> Physical Memory
- With demand paging, physical memory fills up quickly
- When a process faults and memory is full, some page must be swapped out
  - Handling a page fault now requires two disk accesses, not one
- Which page should we replace?
  - Local replacement - replace a page of the faulting process
  - Global replacement - possibly replace the page of another process

Page Replacement Evaluation

- Record a trace of the pages accessed by a process
  - Generate a page trace:
    - e.g. 1, 7, 7, 4, 6, 5, 8, 1, 1, 1, 7, 7, 7, 6, 5, 6
- Simulate the behavior of a page replacement algorithm on the trace and record the number of page faults generated
- Parameters: algorithm, page reference string, # of memory frames
- Fewer faults -> better performance

Page Replacement Algorithms

- A page replacement algorithm picks a page to be paged out and frees up a frame
  - Optimal - the one that leads to the least faults
  - FIFO - First In, First Out
  - LRU - Least Recently Used

First In First Out (FIFO)

- Idea: The oldest page in physical memory is the one selected for replacement
- Extremely simple to implement
    - Keep a list
        - Victims are chosen from the tail
        - New pages are placed on the head
- Performance can be poor
    - FIFO does not consider page usage
        - In the worst case, each page that is paged out could be the one that is referenced next

Least Recently Used (LRU)

- Idea: Replace the page in memory that has not been accessed for the longest time
  - If a page wasn't used recently, then it is unlikely to be used again in the near future
  - If a page was used recently, then it is likely to be used again in the near future
  - Select victim that was used least recently

LRU Example

- Access: 1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3, 6, 1, 2
- 3 frames

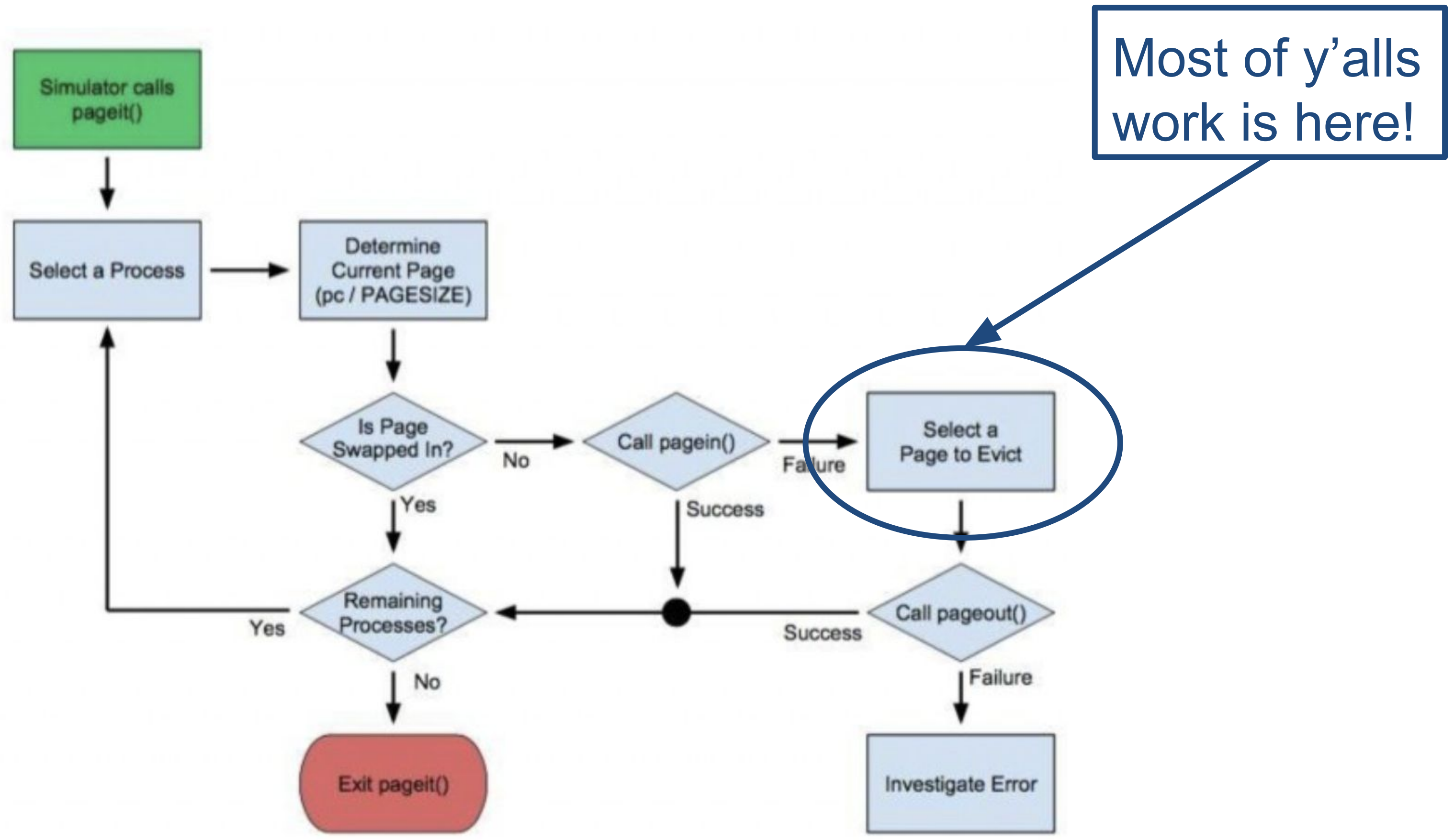
	1	**2**	**3**	**2**	**1**	**5**	**2**	**1**	**6**	**2**	**5**	**6**	**3**	**1**	**3**	**6**	**1**	**2**
	1	1	1	1	1	1	1	1	1	1	5	5	5	1	1	1	1	1
		2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	2
			3	3	3	5	5	5	6	6	6	6	6	6	6	6	6	6
	⊗	⊗	⊗			⊗			⊗		⊗		⊗	⊗				⊗

Nine page faults!

Upcoming PAs

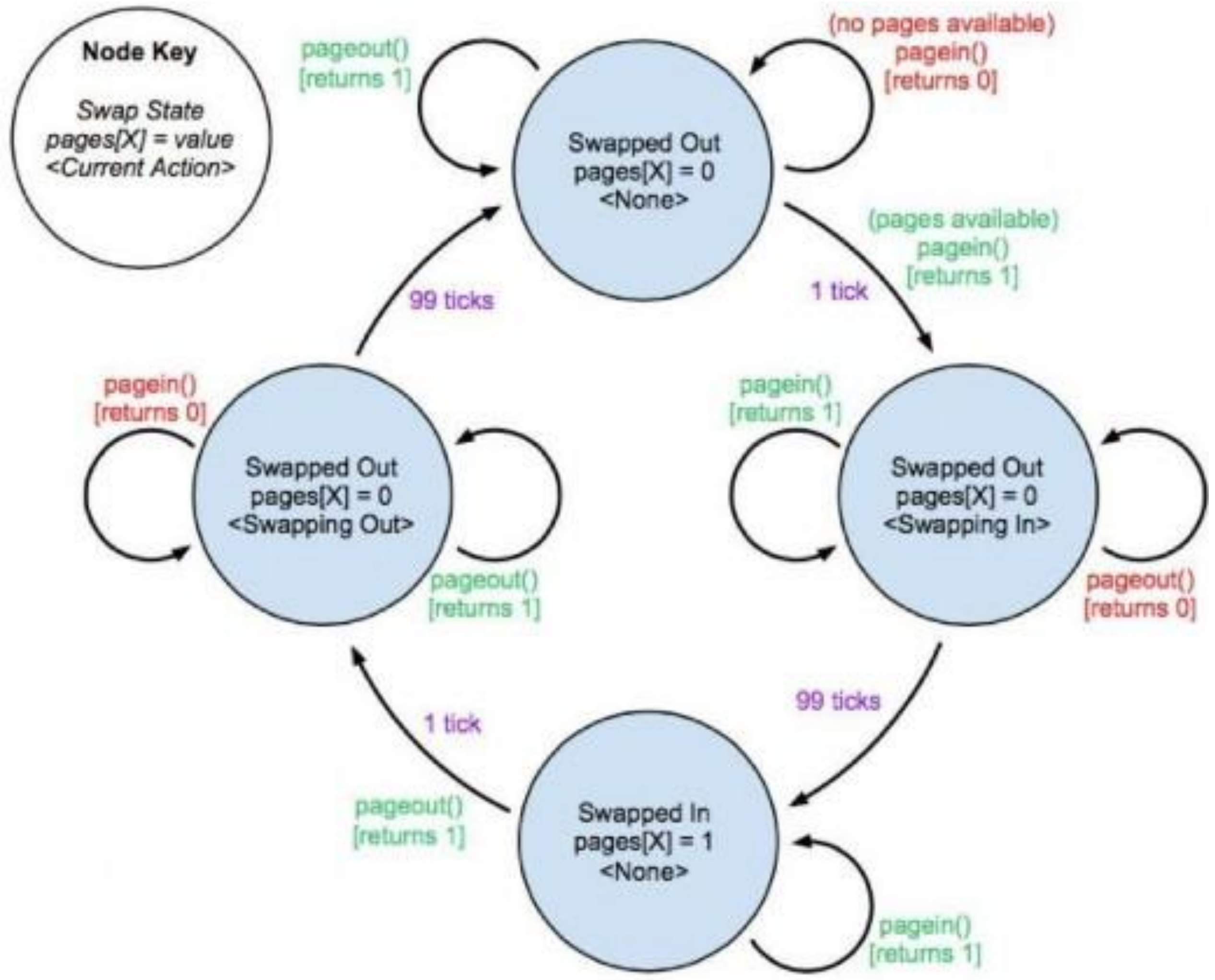
- Goal: Implement a paging strategy which a paging simulator can use to maximize the performance of the memory accesses in a set of predefined programs
  - PA7: LRU
  - PA8: Predictive algorithms

PA7 Diagram



Basic swapping algorithm for single process already provided in pager-basic.c!!

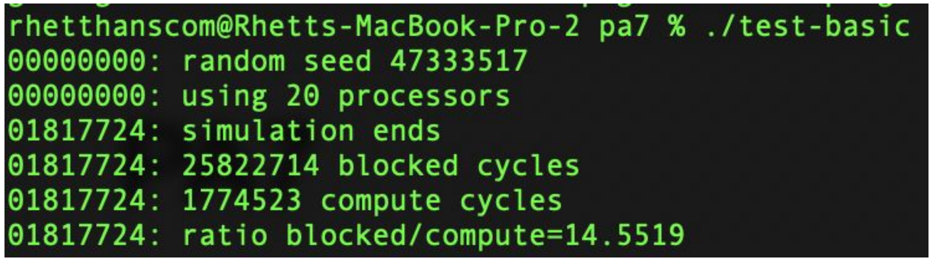
PA7 Diagram


Node Key
Swap State
pages[X] = value
<Current Action>
pageout()
[returns 1]
(no pages available)
pagein()
[returns 0]
Swapped Out
pages[X] = 0
<None>
(pages available)
pagein()
[returns 1]
1 tick
99 ticks
pagein()
[returns 0]
pagein()
[returns 1]
Swapped Out
pages[X] = 0
<Swapping Out>
Swapped Out
pages[X] = 0
<Swapping In>
pageout()
[returns 1]
pageout()
[returns 0]
99 ticks
1 tick
pageout()
[returns 1]
Swapped In
pages[X] = 1
<None>
pagein()
[returns 1]

PA7

- Goal: Implement a LRU paging strategy which a paging simulator can use to improve the performance of the memory accesses in a set of predefined programs
  - pager-basic.c offers a simple, already completed pageit() function which only allows for a single process and page to be in memory at a time
    - Great for seeing/understanding how to work with the data
  - pager-lru.c is where you will do your work

To make and run: **make all**
./test-basic OR ./test-lru



```
rhetthanscom@Rhetts-MacBook-Pro-2 pa7 % ./test-basic
00000000: random seed 47333517
00000000: using 20 processors
01817724: simulation ends
01817724: 25822714 blocked cycles
01817724: 1774523 compute cycles
01817724: ratio blocked/compute=14.5519
```